CHAPTER 21

Creating Forms and Inserting Scripts

In This Chapter

Introducing forms

Understanding form processing

Inserting the FORM

Adding controls

INPUT element

BUTTON element

SELECT, OPTION, and OPTGROUP elements

TEXTAREA element

LABEL element

FIELDSET and LEGEND elements

Form processing options

Dealing with form data

The face of the Web changed dramatically when Web forms became possible. Unfortunately, forms haven't changed much since then. This chapter teaches you about all the basic controls forms can include. It covers how form processing works, what your choices are, and what you can do with the data you collect.

In addition to all the existing form fields you've seen before, HTML 4 introduces a few new form-related elements, including the feature that enables you to link a field to the caption for that field. This chapter discusses all these new elements with examples.

Introducing Forms

Forms are a way of collecting information from your site visitors. What you collect, how you collect it, and how you process it vary from site to site and from application to application.

How does it work?

1. You place data-entry fields on your Web page. These can include text fields, radio buttons, checkboxes, text area fields (for more extensive data entry), and other buttons, such as submit and reset buttons.
2. Your site visitors fill in the fields from their own computers and click a button.

3. The information typed by the visitors is sent over the Internet to your server.
4. Your server takes action on the data.
5. Ideally, you provide your visitors with a screen confirming you have received the data and that some action will be taken.

You can do fancier things, such as sending a confirming message by e-mail, sending yourself a notice that someone is using your form, or entering the data submitted by the visitor into a database.

Understanding Form Processing

Before you start designing your form, consider what you want to do with the data you will be collecting. What can you do with the data you collect from a form? All options fall into one of three categories:

1. Saving the data for further processing
2. Returning information to the visitor
3. Taking other action (such as financial transactions)

Saving the data for further processing

You might want to do this if you require people to register for your site or if you want to collect names for a mailing list. You could store the data in a database to verify the visitor is registered before giving access on subsequent visits. You might run a raffle on your page with a prize randomly awarded to people who sign in to your site. You might require people to sign in before they download your free software or newsletter. In any case, you need to collect certain data and store it on your server — either in a database or in an ASCII file.

Returning information to the visitor

You could perform processing based on the data. For example, you might search through a book list to find books that match the visitor's list of favorite types of literature. If the processing doesn't require any input from the server, then this processing is better performed using client-side processing. If, however, your processing requires access to data that resides on the server, such as a card catalog, a site index, or an inventory database, then you can use the data provided by the visitor to perform server-side processing against existing data.

This is probably the most common and acceptable use of forms on the Web. People aren't as likely to question or refuse providing information when they can get something in return, such as information they are seeking. You need to be reason

able with what kind of information you require your visitors to enter. Most people will not give you a social security number or credit card information unless they are purchasing something.

Taking other action

If someone comes to your site to purchase something, that visitor expects to provide certain types of information, including personal information, payment information, and shipping information. One of the processing decisions you must make is whether you will collect and retain the payment information on the server or whether you will send the information offline to a place where it is less vulnerable to hackers. In either case, you'll collect information using a form or a series of forms, and all your site visitor normally receives back is confirmation the purchase was completed.

Inserting the FORM

There are two parts to the form: The first is the `FORM` element and the second is the fields or controls that accept data. Without a `FORM` element, your form may appear to look fine when you test it, but it will never work.

Form `<FORM>`

Start Tag:	Required
Content:	Inline elements
	Form field elements
End Tag:	Required
Attributes:	`id`, `class`, `lang`, `title`, `style`: previously defined
	`action`: the URL of the script for processing the data or a mailto address if the action is just to mail the data
	`accept_charsets`: list of supported character sets
	`dir`: direction of text
	`name`: name of form for scripting
Attributes:	`accept`: list of MIME types for file upload
	`isindex`: single-line text input control
	`method`: how the data is to be sent to the script
	`enctype`: what types of files will be accepted if one or more of form fields accept file uploads
	`target`: the target name of the frame within the frameset, if your script is a frameset
	`events`: see Chapter 48

The two attributes you see used most frequently are action and method.

```
<FORM method="dosomething.cfm" action="post">
... form fields here ...
</FORM>
```

Action

The `action` attribute tells the form where to send the data it collects. Without an action, your data won't be sent anywhere. Ideally, your action will be the URL of a file containing a script that does something with the data or the URL of a page containing server-side processing.

If you start sniffing through other people's HTML (using View Source in your browser, or Page Source in Netscape 4.x), you will notice that a lot of action attributes don't call scripts. What they do instead is send e-mail, using the `mailto:` method. How does this work? Instead of a URL, you specify `mailto:` followed by your e-mail address. This instructs your browser to send this data to your e-mail address.

Is this a good thing? It depends. It's easy to do. You can have a "functional" form on your Web page in minutes. But does this really do you any good? Not if you expect a high volume of traffic to your site. If a lot of people use your form, you may soon be inundated by data you don't have the capacity to process. If you expect only a few people to complete your form each week, then you may have time to process this data manually.

The bottom line is you should have a real script to process your data if you expect to do anything serious with your form.

Method

The method attribute indicates how you want the data to be sent. Your choices are "get" and "post." When you use the "get" method, you send the data as part of the URL. You are limited to 100 characters with the "get" method. This means that if your form data might approach 100 characters, and one text area could exceed 100 characters if you don't set it up properly, then you cannot use the "get" method.

With the "post" method, no limits exist on the number of characters you can send. Also, the values of all the input fields aren't displayed on the browser's location line, the way they are with the "get" method. Generally speaking, "post" is the preferred method of sending form data to a script.

Have you ever noticed when you do a search on a search engine that the URL in your location window gets very long with strange codes? This is because search engines use `method="get"` (for example, `http://www.lycos.com/srch/?loc=searchbox&ss=957455750%7C2&query=HTML+parameters`).

Adding Controls

Controls are the generic name for all the form fields you can add to your page — the second necessary component of the form. You can add nine types of controls; they do everything from enabling a person completing your form to enter text, to accepting yes or no answers from that person, to enabling that person to choose one or more items from a list you provide. Controls also define the buttons your visitor needs in order to submit data.

You need to know what type of data your form needs to collect before you create controls. Table 21-1 shows what controls can help you collect values for each type of data.

Table 21-1
List of Controls by Type of Data Desired

Type of Data to Be Collected	*What Will Be Displayed on the Form*	*Control to Use*	*Value String Returned to Script*
text data to be processed or evaluated	text entry field, the size of which is defined using the `size` attribute	`INPUT type="text" name="text1"`	`text1="the text entered"`
text data (as lengthy comments)	a large text area field for data entry, the size of which is defined using the `rows` and `cols` attributes	`TEXTAREA name="textarea1"`	`textarea1="very long text string. May include punctuation."`
text data that needs to be protected from view, such as password data	text entry field, but characters typed appear as either asterisks or dots	`INPUT type="password" name="password1"`	`password1="visible text"`

Continued

Table 21-1 *(continued)*

Type of Data to Be Collected	*What Will Be Displayed on the Form*	*Control to Use*	*Value String Returned to Script*
yes/no data (actually returns a value only if the value is "yes")	checkbox, unchecked unless the `checked` attribute is present	`INPUT type="checkbox" name="box1" value="yes"`	`box1="yes"` (returned only if the box is checked)
one value from a short list of choices	radio button, unchecked unless the `checked` attribute is present	`INPUT type="radio" name="set1" value="first"`	`set1="first"` (returned only if the radio button is checked)
one value from a long list of choices	drop-down list of choices	`SELECT name="dropdown1" OPTION value="first"`	`dropdown1`
multiple values from a long list of choices	scrollable list of choices; the visitor must hold down a special key while selecting every choice after the first; list will be the length specified in the `size` attribute; must include multiple attributes	`SELECT name="dropdown1" size="8" multiple OPTION value="first" OPTION value="second"`	`dropdown1="first, second"`
upload a file	text entry field for name of file along with a Browse button to view files on the client computer	`INPUT type="file" accept="application/msword"`	contents of file will be uploaded to server if `enctype` attribute is included on `FORM` element

Type of Data to Be Collected	*What Will Be Displayed on the Form*	*Control to Use*	*Value String Returned to Script*
button	submit form, reset form, or take other action	`INPUT type="submit"` or `INPUT type="reset"` or `INPUT type="image" alt="click to submit"` or `BUTTON`	no data is sent; buttons execute actions but do not send their own data; they just cause the other data to be sent or cleared

INPUT Element

The most general type of control is the `INPUT` element. The `INPUT` element is incredibly flexible. It can be used to create standard text fields, password fields, checkboxes, radio buttons, submit buttons, reset buttons, file input fields, hidden fields, image fields, and general-purpose buttons.

Input `<INPUT>`

Start Tag: Required

Content: Empty

End Tag: Forbidden

Attributes: `id`, `class`, `lang`, `dir`, `title`, `style`: previously defined

`name`: name to be associated with this data when it is sent to the script

`value`: value to be sent for this form field

`alt`: alternate text to be associated with this form field

`checked`: Boolean variable used to indicate a checkbox or radio button should be checked when the page is painted

`disabled`: Boolean variable used to indicate a checkbox or a radio button should be grayed out when the page is painted

`ismap`: use the server-side image map

`align`: vertical or horizontal alignment

`readonly`: Boolean variable that allows a text or password to be seen, but not edited

`size`: indicates the size of the field for text and password fields when the page is painted

`maxlength`: indicates the maximum number of characters that will be accepted as data for text and password fields

`src`: URL of images for fields that have images associated with them

`usemap`: URL of imagemap to be used for client-side imagemaps

`tabindex`: the number of this field in the tabbing order

`accept`: for file uploads, this is the list of MIME types that will be accepted

`accesskey`: the shortcut key associated with this field for accessibility

`events`: see Chapter 48

Type attribute

The `INPUT` element is incredibly flexible, and there is a long list of attributes that are associated with it. Because you will never need all these attributes in one instance of the `INPUT` element, it makes more sense to discuss the `INPUT` element by type. Each of the type values are associated with their own fields.

Text

The text value for the `type` attribute is what you use to define text input fields. The other attributes you might want to include are

- **name.** This is used so the value of the field will have a name when it arrives at the script. If only one field is in the form, a name isn't necessary.
- **size.** The size of the field you want the site visitor to see on the screen when the page is painted.
- **maxlength.** The maximum number of characters you are willing to accept as the value of this field.
- **alt.** Alternate text you want associated with this field.
- **tabindex.** The number you want this field to be in the tabbing order.
- **accesskey.** The keyboard character you want associated with this field as a shortcut.

The `accesskey` attribute enables you to utilize shortcut keys. For example, if you have the word Name in front of a field, and you hold down the Alt key and press on the underlined letter of Name, N, your cursor displays in the form entry field. The Alt key uses the `accesskey` attribute to bring focus to the text box.

Password

When you want a text input field to accept data that you don't want anyone else to see (for example, password data), you can use the password value for the `type` attribute. The other attributes you might want to include are the same as those for the `text type` value.

If you need to have a visitor enter a password, you probably want to include two password fields to enable the visitor to retype the password. You might also want to include a JavaScript to check and make sure the value of both password fields is the same before they can leave the page. This saves you the trouble of doing server processing to confirm the password.

Checkbox

The checkbox value for the `type` attribute enables you to take a yes or no answer from the person completing the form. Note, if a checkbox isn't checked, either by you when you define the checkbox or when the visitor completes the form, then *no data* is sent for this field. This means you won't actually receive *yes* or *no* as an answer to a question. If the question is "May we sell your e-mail address to spam merchants?" and you associate this checkbox with the name *spam*, then you will only receive the field called spam, with whatever value you give it in the value attribute, if the box is checked. If the person does not check the box, the spam field won't be sent to the script at all.

You will want to include at least a few of the following attributes when you define a checkbox:

- **name.** This is used so the script will know what question or statement was checked yes.
- **value.** If you want a particular value associated with this field name, then the value attribute will work because the field is only sent when the value is yes. The name and value pair will always be the same when they are sent. But for some scripts, having the name and value pair is more convenient than having only the field name.
- **alt.** Alternate text you want associated with this field.
- **checked.** If you want the checkbox to be checked when the page is painted by the browser, then you want to include this Boolean attribute.
- **disabled.** If you want the checkbox field to be grayed out when the page is painted, then include this Boolean attribute. You can then enable the field later using JavaScript and the Document Object Model (DOM), as explained in Chapter 47.
- **bindex.** The number you want this field in the tabbing order.
- **accesskey.** The keyboard character you want associated with this field as a shortcut. See the previous note on text `type` fields for more details about this.

What's a Boolean Attribute?

This is in case there are those of you out there who still can't figure out what the heck a Boolean attribute is, as mentioned in the `check` attribute. The term Boolean comes from George Boole, whose claim to fame is a form of symbolic logic, later christened Boolean algebra, in which there are only two ultimate values—true or false.

Another form of Boolean logic uses punctuation and prepositions (the words AND, OR, NOT, and sometimes NEAR as operators, and occasionally quotation marks or brackets) to narrow a search. For example, when performing a search on flax, linen generally appears.

There is a visual representation of Boolean search attributes at: `http://adam.ac.uk/info/boolean.html#bool`.

Radio button

This is similar to a checkbox, in that the site visitor can't make up the value of the field. But unlike the checkbox, a set of radio buttons enable you to give the visitor a choice. Radio buttons are created in sets. When one button of a set is checked, then the other buttons in that set can't be checked. A good example of this is payment information: either you are paying by Visa, Mastercard, or American Express. Radio buttons are perfectly suited to collect this type of data.

You create a set of radio buttons by assigning the same `name` attribute to all the buttons. Then, so you know which button was associated with which value, you assign a unique value to the value attribute of each radio button. For three buttons in a set, the skeletal HTML might look like this:

```
Which of the three colors in the United States flag is your
favorite?<BR>
<RADIO name="flag" value="red" checked> red
<RADIO name="flag" value="white"> white
<RADIO name="flag" value="blue"> blue
```

In the previous example, the button next to the word "red" is checked. One of the three values will be sent with the field name *flag* to the script. The site visitor cannot leave this field blank.

You may also want to include the `alt`, `disabled`, `tabindex`, and `accesskey` attributes, which are defined in the previous "Checkbox" section.

File

You can let site visitors upload files to your server, such as images or resumes, using the file value of the `type` attribute. This is actually a bit trickier than the other types of the `INPUT` element because you must be sure to include the `enctype`

attribute for the FORM element or it won't work. The accept attribute of the INPUT element limits the MIME types your server accepts. Telling people what types of files they can upload is a good idea; then they won't get unhelpful messages from the server and think your form is broken.

Uploading files with the INPUT element doesn't work with all servers. Check with your system administrator to see whether it will work on your server, but it generally works with NT servers.

Uploading files is easy for visitors if they are using any of the earlier version browsers, except for Internet Explorer 3. IE 3 has a bug and won't display the Browse button next to the text entry field. This means visitors must type in the exact path name of the file on their computers for the upload to work. Of course, you can download a patch from the Microsoft site, but this is little solace to a visitor who can't figure out how to use your form.

The following HTML shows the essentials for creating a form that accepts files for uploads. In this example, the Web page is for an employment agency. It only accepts resumes in Microsoft Word format. The get method will not work with file uploads.

```
<FORM action="upload-resume.cgi" method="post" enctype=
"multipart/form-data">
Upload your resume by finding it on your local hard drive,
using the Browse button below. Remember that your resume must
be in Microsoft Word format, with a file extension of .doc.<BR>
Full name as it appears on the resume: <INPUT type="text"
name="full_name" size="30" maxlength="50"><BR>
Social security number: <INPUT type="text" name="ssn" size="15"
maxlength="11"><BR>
Resume file: <INPUT type="file" name="resume" size="30" accept=
"application/msword"><BR>
<INPUT type="submit" value="Send Your Resume Now!">
<INPUT type="reset" value="Reset Form">
</FORM>
```

Notice in the previous form, no special action is required to create the Browse button for the file upload. The browser creates this automatically when it sees the input type is file.

Hidden

What is the point of putting a hidden form field on your page? After all, who can fill in a field if it's hidden? Actually, a *hidden field* isn't really a form field, but a way to pass data from one page to another. For many reasons, you may want to pass data from one page of your site to another. For example, you may have already asked for information on a previous page that you don't want to ask for again. By using a hidden field, you can pass that data into your script without the visitor knowing. You might also include a hidden field to ensure the script is really receiving data from

your form, rather than from another page (for example, someone trying to break into your system). Or, if a script is multipurpose, you might include a hidden field to tell the script which action you want it to take.

Your basic hidden field looks like this:

```
<INPUT type="hidden" name="this_page" value="page1.html">
```

The hidden field doesn't show up on your page at all. You would have to view the source for the page to know the field is exists. Also, the hidden field must be placed between the `FORM` element's begin and end tags or the field won't be sent with the rest of the data.

Submit or reset

Input types of submit or reset create buttons that either submit the form data to the `action` attribute of the `FORM` element, or reset all the fields to the values you assigned using the `value` attributes.

You can specify what text appears in your button by setting the value attribute. The following code shows three different ways to create a submit button that says "Purchase!" on it. The GIF file, called *purchase-image.gif*, contains an image that says "Purchase!"

```
<INPUT type="submit" value="Purchase!">
<INPUT type="image" src="images/purchase-image.gif"
alt="Purchase!">
<BUTTON type="submit">Purchase!<IMG src="images/purchase-
image.gif" alt="Purchase!"></BUTTON>
```

Image

The *image value* of the `type` attribute enables you to create a submit button that uses an image for the button. This can be more visually appealing, but it can also be more confusing. People are used to buttons on the Web looking a certain way. Unless you make your image look like a button, you risk having people complete your form and then not understand how to submit the data.

Creating an image button is easy:

```
<INPUT type="image" src="images/accept.gif" alt="Accept">
```

Button

The button value of the type attribute is yet another way to create a button on your page. In case you lost count, this is the fourth way to do this using the `INPUT` element. The *button value* of the type attribute is a way to execute client-side scripts, rather than the standard submit or reset actions of the other buttons.

Client-side scripts usually use JavaScript or VBScript. One type of button you might want to include checks to make sure a password meets your criteria. By including a button labeled "Check Password" next to two password entry fields, you can perform this type of verification on the client's machine even before you send the data across the network.

The following HTML shows how you might do this. The JavaScript in this example is called `TestPasswords()`. It confirms that the passwords in both password fields match each other and are of a certain length, don't use the same letter repeatedly, and a few other things. If the password doesn't pass these tests, a message is returned to the visitor with instructions on which rules weren't met.

```
<FORM action="password-change.cfm" method="post">
Enter your old password here: <INPUT type="password" name="old-
password" size="12" maxlength="12"><BR>
Enter your new password here: <INPUT type="password" name="new-
pwd1" size="12" maxlength="12"><BR>
Please re-enter your new password: <INPUT type="password"
name="new-pwd2" size="12" maxlength="12"><BR>
<INPUT type="button" value="Check new passwords" onclick=
"TestPasswords()">
<INPUT type="reset">
<INPUT type="submit" value="Submit passwords for change">
</FORM>
```

BUTTON Element

To add a bit more confusion to the whole discussion of buttons, in addition to the four ways you can create buttons using the `INPUT` element, the W3C has created the `BUTTON` element. The `BUTTON` element is supposed to give you more flexibility in creating buttons. It is hoped the browsers will implement the `BUTTON` element in such a way that both text buttons and image buttons actually recess when clicked, the way most menu buttons do in normal desktop applications.

Button `<BUTTON>`

Start Tag: Required

Content: Inline elements

End Tag: Required

Attributes: `id`, `class`, `lang`, `dir`, `title`, `style`: previously defined
`name`: name to be associated with the data sent to the script (optional)

`value`: value sent to the script (optional)

`type`: submit, reset, or button

`disabled`: for use with JavaScript and the Document Object Model

`tabindex`: a number assigning the tabbing order through the form

`accesskey`: shortcut key to access button for accessibility purposes

`events`: see Chapter 48

Why would you ever use the `BUTTON` element when you can create buttons in four other ways, including buttons with images on them? The `BUTTON` element is far more flexible than any of the other ways of creating buttons. Additionally, the `INPUT` element's variations of buttons doesn't actually depress on the page the way people expect buttons to depress when they are clicked. HTML 4 specifies browsers should support the `BUTTON` element by having a button take action when it is clicked.

The `BUTTON` element has several attributes, but you rarely need more than the type attribute. Defining a button with the `BUTTON` element is different from defining a button with the `INPUT` element because the `BUTTON` element takes content. It also requires an end tag. You include an image as the button in the `BUTTON` element by including the `IMG` element between the start tag and the end tag. You should also include text in the content so that if for some reason the image doesn't load, text is available to indicate what the button does.

Two examples of buttons created with the `BUTTON` element follow:

```
<BUTTON type="submit">Click here to submit!<IMG
src="images/submit.gif"></BUTTON>
<BUTTON type="button" onclick="ValidateForm()">Validate form
before submitting</BUTTON>
```

SELECT, OPTION, and OPTGROUP Elements

The standard drop-down list, known in HTML as the *select list*, is created with the `SELECT` element and the `OPTION` element. HTML 4 introduces the `OPTGROUP` element, which enables you to group `OPTION` elements so your standard drop-down list looks more like a directory listing than like the drop-down lists you are used to seeing, with a single, long listing of all the options.

Figure 21-1 shows two types of select lists. The first is the standard select list with a drop-down box. You can select one item from a list that can be as long as you wish. The second shows a select list that actually looks like a scrollable list. Notice eight lines are visible. In this list, you can choose more than one from the eight items. As of publication, none of the major browsers supported the `OPTGROUP` element, so you have to see how this is supported on your own desktop.

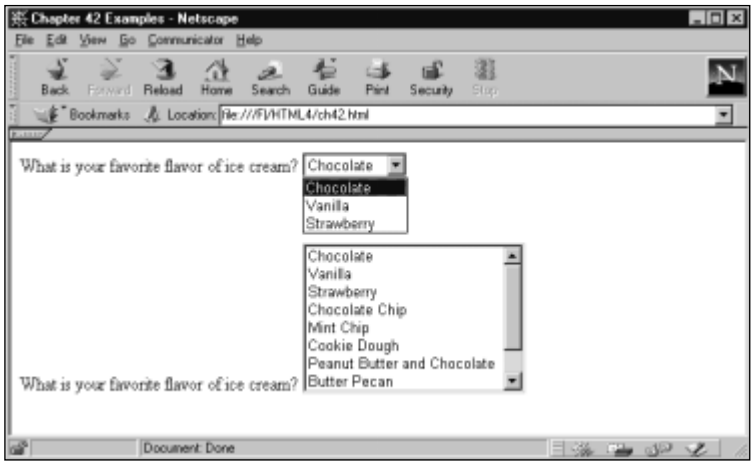

Figure 21-1: Two different types of select lists

SELECT element

The `SELECT` element is the structure in which you build a select list. A select list should have a name, but doesn't generally demand any of the other attributes available to it. A `SELECT` element must have at least one `OPTION` element.

Select `<SELECT>`

Start Tag: Required

Content: `OPTGROUP`, `OPTION`: elements text

End Tag: Required

Attributes: `id`, `class`, `lang`, `dir`, `title`, `style`: previously defined

`name`: name associated with the value or values returned to the script

`size`: if this is to be a scrollable list instead of a drop-down list, you must specify the number of rows to be visible with the size attribute

`multiple`: Boolean value that indicates more than one value may be selected; must be used in conjunction with the size attribute

`disabled`: for use with JavaScript and the DOM

`tabindex`: indicates tabbing order in form; a number

`events`: see Chapter 48

Two examples of the `SELECT` element in action appear later in this section.

OPTION element

For each item on the select list you want your visitors to be able to select, you need an instance of the OPTION element.

Option <OPTION>

Start Tag: Required

Content: Text

End Tag: Optional

Attributes: id, class, lang, dir, title, style: previously defined

selected: indicates the drop-down list or scrollable list will begin with this item in focus or highlighted

disabled: for use with JavaScript and the DOM

label: to be sent as the value of this selection; should be used if the contents of the OPTION element are too long to be useful for processing

value: the initial value of the choice; normally the same as the contents of the OPTION element

The OPTION element is straightforward. For each item you want in your drop-down list, you create an OPTION element. In the example in Figure 21-1, the following HTML was used to create the drop-down list.

```
What is your favorite flavor of ice cream?
<SELECT name="test1">
  <OPTION>Chocolate
  <OPTION>Vanilla
  <OPTION>Strawberry
</SELECT>
```

Using attributes

If one of the flavors had been called "Very Berry Razzamatazz Strawberry" and you didn't want to send all this information to the script, you use the following OPTION element (in addition to the previous ones).

```
<OPTION label="Strawberry">Very Berry Razzamatazz Strawberry
```

If you don't include a label attribute, the contents of the OPTION element will be sent as the value selected. Notice that even though an end tag is optional, it isn't generally used.

If you wanted the select list to start off with the word *Vanilla* in view, instead of just having the first OPTION element in the list in view, you could have used the selected attribute.

```
<OPTION selected>Vanilla
```

OPTGROUP element

The OPTGROUP, once it is fully implemented, will make select lists much more powerful and flexible. The OPTGROUP is intended to work like the drop-down menus you see at the top of most applications. It is a way of organizing your select lists.

Option Group <OPTGROUP>

Start Tag: Required

Content: OPTION elements

End Tag: Required

Attributes: id, class, lang, dir, title, style: previously defined

disabled: for use with JavaScript and the DOM

label: to appear in the select list as the title of this group

HTML 4 does not allow for nested OPTGROUP elements, but it won't be surprising to see the major browsers implement it anyway. Currently, none of the major browsers implement the OPTGROUP element.

Unlike the OPTION element, where the label attribute is optional and is used only if the contents of the element are too long to send and process, the label attribute on the OPTGROUP element is necessary if you want the group heading to appear in the list. Also, unlike the OPTION element, the visitor can't select the label of the OPTGROUP as the value to send to the script. The OPTGROUP only leads to a list of options. All the selectable options must be in OPTION elements.

For example, if you wanted to create a select list with the following groups:

"What is your favorite frozen dessert?"

Ice Cream

Chocolate
Vanilla
Strawberry
Chocolate Chip
Mint Chip
Cookie Dough
Peanut Butter and Chocolate

Sorbet

Raspberry
Lemon
Lime

Frozen Yogurt

Vanilla
Chocolate
Peanut Butter
Raspberry

The HTML would be:

```
What is your favorite frozen dessert?
<SELECT name="test3" multiple size="8">
  <OPTGROUP label="Ice Cream">
       <OPTION>Chocolate
       <OPTION>Vanilla
       <OPTION>Strawberry
       <OPTION>Chocolate Chip
       <OPTION>Mint Chip
       <OPTION>Cookie Dough
       <OPTION>Peanut Butter and Chocolate
  </OPTGROUP>
  <OPTGROUP label="Sorbet">
       <OPTION>Raspberry
       <OPTION>Lemon
       <OPTION>Lime
  </OPTGROUP>
  <OPTGROUP label="Frozen Yogurt">
       <OPTION>Vanilla
       <OPTION>Chocolate
       <OPTION>Peanut Butter
       <OPTION>Raspberry
  </OPTGROUP>
</SELECT>
```

TEXTAREA Element

The TEXTAREA element is a way to accept free-form text from your visitors. If you want to accept suggestions or other types of lengthy text, this is the control to use.

Text Area <TEXTAREA>

Start Tag: Required

Content: Initial text to appear in TEXTAREA

End Tag: Required

Attributes: id, class, lang, dir, title, style: previously defined

name: name associated with the value or values returned to the script

rows: number of rows of text permitted

cols: number of columns of text visible

disabled: for use with JavaScript and the DOM

readonly: shows contents of TEXTAREA element, but doesn't allow editing

tabindex: number indicating tab order in form

accesskey: shortcut key for accessibility

events: see Chapter 48

Text areas are easy to create. To define one that has 4 visible rows that are 60 columns wide, with the initial text "Enter your comments here," you would use the following HTML:

```
<TEXTAREA name="comments1" rows="4" cols="60">Enter your
comments here.</TEXTAREA>
```

LABEL Element

The LABEL element enables you to attach text to the control. When you create most form controls, such as text fields, select lists, or checkboxes, there is some text, such as a question, that isn't attached in any way to the actual control that answers that question.

Why would you want to attach text to a control? So site visitors can click the text, as well as the control (such as a tiny checkbox). The text associated with a control may not be immediately adjacent to a control—when using tables for formatting, for example, associating text with a control is a little bit tricky.

Label <LABEL>

Start Tag: Required

Content: Text

End Tag: Required

Attributes: id, class, lang, dir, title, style: previously defined

for: indicates the ID value of the control with which it is associated

accesskey: shortcut key for accessibility

events: see Chapter 48

The LABEL element (not to be confused with the label attribute of the OPTION and OPTGROUP elements) is attached to the text you want associated with a control. Here are the steps involved in assigning text to a control:

1. Use the id attribute of the form control to assign an ID to the form field.
2. Place the text you want associated with the form control within a LABEL element.
3. Assign the for attribute of the LABEL element to match the value of the id in the form control.

Here is a relatively complex example using tables for formatting a form.

```
<FORM action="do-something.cgi" method="post">
<TABLE>
<TR>
<TD align="right"><LABEL for="fname">Please enter your first
name</LABEL>
<TD><INPUT type="text" name="first_name" id="fname">
<TR>
<TD align="right"><LABEL for="lname">Please enter your last
name</LABEL>
<TD><INPUT type="text" name="last_name" id="lname">
<TR>
<TD align="right"><LABEL for="spam">May we sell your personal
data to spammers?</LABEL>
<TD><INPUT type="checkbox" name="spam" value="yes" id="spam">
<TR>
<TD align="right><LABEL for="comments">Please enter any
comments you have about our site</LABEL>
<TD><TEXTAREA name="comments" id="comments" rows="4"
columns="40">Enter your comments here</TEXTAREA>
<TR>
<TD><INPUT type="submit" value="Send this now">
<INPUT type="reset" value="Clear">
</TABLE>
</FORM>
```

Notice that the `id` and the `name` can be the same, but that isn't always necessary.

FIELDSET and LEGEND Elements

The `FIELDSET` and `LEGEND` elements work together. The `FIELDSET` is strictly a structural element; its presence or absence doesn't affect the rendering of your page. The `LEGEND` element is an added convenience for visitors "seeing" your Web page with their ears.

Fieldset `<FIELDSET>`

Start Tag:	Required
Content:	`LEGEND` element elements
End Tag:	Required
Attributes:	`id`, `class`, `lang`, `dir`, `title`, `style`: previously defined

Legend <LEGEND>

Start Tag: Required

Content: Text of the legend

End Tag: Required

Attributes: id, class, lang, dir, title, style: previously defined

accesskey: shortcut key for accessibility

Neither of these elements would be difficult to implement. The FIELDSET element should surround related fields so they are grouped. The LEGEND element immediately follows the FIELDSET element with a descriptive title for the FIELDSET.

Form Processing Options

Today, more options for processing your data are on the server than this book can cover. Just a few of the options available to you include:

- **Common Gateway Interface Scripts (CGI scripts).** This is the oldest type of server-side processing. You can write your program/script in Perl or C, or by using a shell script. This is still the most common type of script running on UNIX servers.
- **Server-side JavaScript.** JavaScript is a widely known scripting language. It can, in fact, be used for full-scale, object-oriented programs. Netscape has introduced servers that include server-side JavaScript processing.
- **Server-side Java.** Java has almost caught up with the hype associated with it when it was first introduced. You can use Java on your server to perform many form-processing functions. If you know Java, and if your Web server supports server-side Java (Microsoft Internet Information Server [IIS] does not), then this is a good option. If you don't know Java, much easier ways exist to accomplish server-side processing of forms.
- **Cold Fusion.** Cold Fusion, by Allaire, is the easiest way to add server-side processing to your pages. In only minutes, you can have a form on your page that collects data into a database and sends out confirmation mail. New features are constantly being added to this product to make it even more useful.
- **Active Server Pages (ASP).** ASP was created by Microsoft, and can be created using JavaScript, PERL, and C++, but true ASP is done using VBScript. Your server must be running the Microsoft IIS operating system, version 3.0 or higher, and then implementing ASP on your site is simple. Front Page 98, the Microsoft Web authoring tool, makes this even easier.

Dealing with Form Data

What is the best way to handle form data? With a script or program. Ideally, you will know how you are planning to use each piece of data you request before you request it. Then, you will have a program or script in place that acts on the data when you first publish your form.

When you create your form, you need to give some thought to how each piece of returned data will be used. If you don't plan to look at or respond to comments, don't ask for them. You will probably find that if you have multiple-word responses to questions in select lists, working with shortened versions of these responses using the label attribute in the `OPTION` element will be easier, as shown here:

```
<OPTION label="girl colors">Red, pink, fuchsia, yellow, or
orange
```

Where the question might have been: "In what colors do you prefer to dress your daughter?"

If you make use of checkboxes or radio buttons, remember that a value is only sent when the checkbox or radio button is checked. This means even from a long set of checkboxes, it is possible that no values will be returned. Your script must be able to deal with the absence of data.

What if you don't know how to write a script, but you still want to collect data? You can use a `mailto:` script.

Mailto:

With your e-mail address as the action of the script, `mailto:` sends the field names (as set by you in the name attributes) and their values to you via e-mail. The problem with this technique is, while you have the data, you don't have the data in any particularly useful form. All that is returned to you is a list with each line containing a field name and the value entered for that field.

```
last_name = Miller
first_name = John
email = miller@somewhere.org
color = blue
flavor = Mint Chip
music = Meatloaf
```

Even worse than having the data returned in a difficult-to-process format, the `mailto:` script doesn't always return the fields in the order they were completed.

For these reasons, `mailto:` should be used only when you really do plan for the data to go no further than your mailbox.

From Here

Jump to Chapter 45 and learn about accessing external databases.

Proceed to Chapter 22 and learn about using frames

Summary

Forms enable you to take input from your site visitor for processing on your server. If you want to add a site search, sell anything, or take a survey, you must learn how to put form fields on your page. A lot of flexibility exists in creating forms. Also, HTML 4 specifies some new elements, which give you more control over buttons and enable you to group your fields. And more accessibility features are in HTML 4, such as the `tabindex` attribute and the `accesskey` attribute.

The most important thing to know about forms is that you must have plans for processing your data; otherwise you are wasting your visitors' time in collecting data. Before you put up a form, be sure you have plans to use the data wisely.

✦ ✦ ✦